DP-420 Q&As

Designing and Implementing Cloud-Native Applications Using Microsoft Azure Cosmos DB

Pass Microsoft DP-420 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.geekcert.com/dp-420.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Microsoft Official Exam Center

- **Instant Download** After Purchase
- **100% Money Back** Guarantee
- **365 Days** Free Update
- **800,000+** Satisfied Customers

QUESTION 1

You have an Azure Cosmos DB database.

You plan to create a new container named container1 that will store product data and product category data and will primarily support read requests.

You need to configure a partition key for container1. The solution must meet the following requirements:

Minimize the size of the partition.

Minimize maintenance effort.

Which two characteristics should you prioritize? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. unique

B. high cardinality

C. low cardinality

D. static

Correct Answer: BD

Explanation:

B: For all containers, your partition key should:

*

Have a high cardinality. In other words, the property should have a wide range of possible values.

*

Etc.

D: Be a property that has a value, which doesn\\'t change. If a property is your partition key, you can\\'t update that property\\'s value.

Reference: https://learn.microsoft.com/en-us/azure/cosmos-db/partitioning-overview

QUESTION 2

You are implementing an Azure Data Factory data flow that will use an Azure Cosmos DB (SQL API) sink to write a dataset. The data flow will use 2,000 Apache Spark partitions. You need to ensure that the ingestion from each Spark partition is balanced to optimize throughput.

Which sink setting should you configure?

A. Throughput

B. Write throughput budget

C. Batch size

D. Collection action

Correct Answer: C

Batch size: An integer that represents how many objects are being written to Cosmos DB collection in each batch. Usually, starting with the default batch size is sufficient. To further tune this value, note:

Cosmos DB limits single request\\'s size to 2MB. The formula is "Request Size = Single Document Size * Batch Size". If you hit error saying "Request size is too large", reduce the batch size value.

The larger the batch size, the better throughput the service can achieve, while make sure you allocate enough RUs to empower your workload.

Incorrect Answers:

A: Throughput: Set an optional value for the number of RUs you\\'d like to apply to your CosmosDB collection for each execution of this data flow. Minimum is 400.

B: Write throughput budget: An integer that represents the RUs you want to allocate for this Data Flow write operation, out of the total throughput allocated to the collection.

D: Collection action: Determines whether to recreate the destination collection prior to writing.

None: No action will be done to the collection. Recreate: The collection will get dropped and recreated

Reference: https://docs.microsoft.com/en-us/azure/data-factory/connector-azure-cosmos-db

QUESTION 3

HOTSPOT

You have three containers in an Azure Cosmos DB Core (SQL) API account as shown in the following table.

Name	Database	Time to Live
cn1	db1	On (no default)
cn2	db1	Off
cn3	db1	On (no default)

You have the following Azure functions:

1.

A function named Fn1 that reads the change feed of cn1

2.

A function named Fn2 that reads the change feed of cn2

3.

A function named Fn3 that reads the change feed of cn3 You perform the following actions:

1.

Delete an item named item1 from cn1.

2.

Update an item named item2 in cn2.

3.

For an item named item3 in cn3, update the item time to live to 3,600 seconds.

For each of the following statements, select Yes if the statement is true. Otherwise, select No.

NOTE: Each correct selection is worth one point.

Hot Area:

Answer Area

Statements	Yes	No
Fn1 will receive item1 from the change feed	○	○
Fn2 can check the `_etag` of item2 to see whether the item is an update or an insert	○	○
Fn3 will receive item3 from the change feed	○	○

Correct Answer:

Answer Area

Statements	Yes	No
Fn1 will receive item1 from the change feed	○	●
Fn2 can check the `_etag` of item2 to see whether the item is an update or an insert	○	●
Fn3 will receive item3 from the change feed	●	○

Box 1: No

Azure Cosmos DB\\'s change feed is a great choice as a central data store in event sourcing architectures where all data ingestion is modeled as writes (no updates or deletes).

Note: The change feed does not capture deletes. If you delete an item from your container, it is also removed from the change feed. The most common method of handling this is adding a soft marker on the items that are being deleted. You

can add a property called "deleted" and set it to "true" at the time of deletion. This document update will show up in the change feed. You can set a TTL on this item so that it can be automatically deleted later.

Box 2: No

The _etag format is internal and you should not take dependency on it, because it can change anytime.

Box 3: Yes

Change feed support in Azure Cosmos DB works by listening to an Azure Cosmos container for any changes.

Reference:

https://docs.microsoft.com/en-us/azure/cosmos-db/sql/change-feed-design-patterns

https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed

QUESTION 4

You have a database named db1 in an Azure Cosmos DB for NoSQL

You are designing an application that will use dbl.

In db1, you are creating a new container named coll1 that will store in coll1.

The following is a sample of a document that will be stored in coll1.

```
{
    "customerId" : "bba6fe24-6d97-4935-8d58-36baa4b8a0e1",
    "orderId" : "9d7816e6-f401-42ba-ad05-0e03de35c0b8",
    "orderDate" : "2021-09-29",
    "orderDetails" : []
}
```

The application will have the following characteristics:

1.

New orders will be created frequently by different customers.

2.

Customers will often view their past order history.

You need to select the partition key value for coll1 to support the application. The solution must minimize costs. To what should you set the partition key?

A. id

B. customerId

C. orderDate

D. orderId

Correct Answer: B

Based on the characteristics of the application and the provided document structure, the most suitable partition key value for coll1 in the given scenario would be the customerId, Option B. The application frequently creates new orders by different customers and customers often view their past order history. Using customerId as the partition key would ensure that all orders associated with a particular customer are stored in the same partition. This enables efficient querying of past order history for a specific customer and reduces cross-partition queries, resulting in lower costs and improved performance. a partition key is a JSON property (or path) within your documents that is used by Azure Cosmos DB to distribute data among multiple partitions3. A partition key should have a high cardinality, which means it should have many distinct values, such as hundreds or thousands1. A partition key should also align with the most common query patterns of your application, so that you can efficiently retrieve data by using the partition key value1. Based on these criteria, one possible partition key that you could use for coll1 is B:customerId.

This partition key has the following advantages: It has a high cardinality, as each customer will have a unique ID3. It aligns with the query patterns of the application, as customers will often view their past order history3. It minimizes costs, as it reduces the number of cross-partition queries and optimizes the storage and throughput utilization1. This partition key also has some limitations, such as: It may not be optimal for scenarios where orders need to be queried independently from customers or aggregated by date or other criteria3. It may result in hot partitions or throttling if some customers create orders more frequently than others or have more data than others1. It may not support transactions across multiple customers, as transactions are scoped to a single logical partition2. Depending on your specific use case and requirements, you may need to adjust this partition key or choose a different one. For example, you could use a synthetic partition key that concatenates multiple properties of an item2, or you could use a partition key with a random or pre-calculated suffix to distribute the workload more evenly2.

QUESTION 5

You have a container named container1 in an Azure Cosmos DB for NoSQL account named account1 that is set to the session default consistency level. The average size of an item in container1 is 20 KB.

You have an application named App1 that uses the Azure Cosmos DB SDK and performs a point read on the same set of items in container1 every minute.

You need to minimize the consumption of the request units (RUs) associated to the reads by App1.

What should you do?

A. In account1, change the default consistency level to bounded staleness.

B. In App1, change the consistency level of read requests to consistent prefix.

C. In account1, provision a dedicated gateway and integrated cache

D. In App1, modify the connection policy settings.

Correct Answer: B

The cost of a point read for a 1 KB item is 1 RU. The cost of other operations depends on factors such as item size,

indexing policy, consistency level, and query complexity1. To minimize the consumption of RUs, you can optimize these

factors according to your application needs.

For your scenario, one possible way to minimize the consumption of RUs associated to the reads by App1 is to change the consistency level of read requests to consistent prefix. Consistent prefix is a lower consistency level than session,

which is the default consistency level for Azure Cosmos DB. Lower consistency levels consume fewer RUs than higher consistency levels2. Consistent prefix guarantees that reads never see out-of-order writes and that monotonic reads are

preserved1. This may be suitable for your application if you can tolerate some eventual consistency.